PHP Example Programs:

Syntax for single-line comments:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// This is a single-line comment  
  
# This is also a single-line comment  
?>  
  
</body>  
</html>

Syntax for multiple-line comments:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
/\*  
This is a multiple-lines comment block  
that spans over multiple  
lines  
\*/  
?>  
  
</body>  
</html>

Using comments to leave out parts of the code:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// You can also use comments to leave out parts of a code line  
$x = 5 /\* + 15 \*/ + 5;  
echo $x;  
?>  
  
</body>  
</html>

Variables are "containers" for storing information.

**Creating (Declaring) PHP Variables**

In PHP, a variable starts with the $ sign, followed by the name of the variable:

**Example**

<?php  
$txt = "Hello world!";  
$x = 5;  
$y = 10.5;  
?>

## PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume).

Rules for PHP variables:

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive ($age and $AGE are two different variables)

Remember that PHP variable names are case-sensitive!

## Output Variables

The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:

### Example

<?php  
$txt = "W3Schools.com";  
echo "I love $txt!";  
?>

The following example will produce the same output as the example above:

### Example

<?php  
$txt = "W3Schools.com";  
echo "I love " . $txt . "!";  
?>

The following example will output the sum of two variables:

### Example

<?php  
$x = 5;  
$y = 4;  
echo $x + $y;  
?>

**Note:** You will learn more about the echo statement and how to output data to the screen in the next chapter.

## PHP is a Loosely Typed Language

## PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

## The PHP echo Statement

The echo statement can be used with or without parentheses: echo or echo().

**Display Text**

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

### Example

<?php  
echo "<h2>PHP is Fun!</h2>";  
echo "Hello world!<br>";  
echo "I'm about to learn PHP!<br>";  
echo "This ", "string ", "was ", "made ", "with multiple parameters.";  
?>

**Display Variables**

The following example shows how to output text and variables with the echo statement:

### Example

<?php  
$txt1 = "Learn PHP";  
$txt2 = "W3Schools.com";  
$x = 5;  
$y = 4;  
  
echo "<h2>" . $txt1 . "</h2>";  
echo "Study PHP at " . $txt2 . "<br>";  
echo $x + $y;  
?>

## The PHP print Statement

The print statement can be used with or without parentheses: print or print().

**Display Text**

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

### Example

<?php  
print "<h2>PHP is Fun!</h2>";  
print "Hello world!<br>";  
print "I'm about to learn PHP!";  
?>

**Display Variables**

The following example shows how to output text and variables with the print statement:

### Example

<?php  
$txt1 = "Learn PHP";  
$txt2 = "W3Schools.com";  
$x = 5;  
$y = 4;  
  
print "<h2>" . $txt1 . "</h2>";  
print "Study PHP at " . $txt2 . "<br>";  
print $x + $y;  
?>

**PHP Data Types**

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

* String
* Integer
* Float (floating point numbers - also called double)
* Boolean
* Array
* Object
* NULL
* Resource

**PHP String**

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

**Example**

<?php   
$x = "Hello world!";  
$y = 'Hello world!';  
  
echo $x;  
echo "<br>";   
echo $y;  
?>

**PHP Integer**

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

* An integer must have at least one digit
* An integer must not have a decimal point
* An integer can be either positive or negative
* Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example $x is an integer. The PHP var\_dump() function returns the data type and value:

**Example**

<?php   
$x = 5985;  
var\_dump($x);  
?>

**PHP Float**

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example $x is a float. The PHP var\_dump() function returns the data type and value:

**Example**

<?php   
$x = 10.365;  
var\_dump($x);  
?>

**PHP Boolean**

A Boolean represents two possible states: TRUE or FALSE.

$x = true;  
$y = false;

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

**PHP Array**

An array stores multiple values in one single variable.

In the following example $cars is an array. The PHP var\_dump() function returns the data type and value:

**Example**

<?php   
$cars = array("Volvo","BMW","Toyota");  
var\_dump($cars);  
?>

You will learn a lot more about arrays in later chapters of this tutorial.

**PHP Object**

Classes and objects are the two main aspects of object-oriented programming.

A class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Let's assume we have a class named Car. A Car can have properties like model, color, etc. We can define variables like $model, $color, and so on, to hold the values of these properties.

When the individual objects (Volvo, BMW, Toyota, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

If you create a \_\_construct() function, PHP will automatically call this function when you create an object from a class.

**Example**

<?php  
class Car {  
  public $color;  
  public $model;  
  public function \_\_construct($color, $model) {  
    $this->color = $color;  
    $this->model = $model;  
  }  
  public function message() {  
    return "My car is a " . $this->color . " " . $this->model . "!";  
  }  
}  
  
$myCar = new Car("black", "Volvo");  
echo $myCar -> message();  
echo "<br>";  
$myCar = new Car("red", "Toyota");  
echo $myCar -> message();  
?>

**PHP NULL Value**

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

**Tip:** If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:

**Example**

<?php  
$x = "Hello world!";  
$x = null;  
var\_dump($x);  
?>

**PHP Resource**

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.

A common example of using the resource data type is a database call.

We will not talk about the resource type here, since it is an advanced topic.

# PHP Strings

A string is a sequence of characters, like "Hello world!".

## PHP String Functions

In this chapter we will look at some commonly used functions to manipulate strings.

## strlen() - Return the Length of a String

The PHP strlen() function returns the length of a string.

### Example

Return the length of the string "Hello world!":

<?php  
echo strlen("Hello world!"); // outputs 12  
?>

## str\_word\_count() - Count Words in a String

The PHP str\_word\_count() function counts the number of words in a string.

### Example

Count the number of word in the string "Hello world!":

<?php  
echo str\_word\_count("Hello world!"); // outputs 2  
?>

## strrev() - Reverse a String

The PHP strrev() function reverses a string.

### Example

Reverse the string "Hello world!":

<?php  
echo strrev("Hello world!"); // outputs !dlrow olleH  
?>

## strpos() - Search For a Text Within a String

The PHP strpos() function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.

### Example

Search for the text "world" in the string "Hello world!":

<?php  
echo strpos("Hello world!", "world"); // outputs 6  
?>

**Tip:** The first character position in a string is 0 (not 1).

## str\_replace() - Replace Text Within a String

The PHP str\_replace() function replaces some characters with some other characters in a string.

### Example

Replace the text "world" with "Dolly":

<?php  
echo str\_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!  
?>

**PHP Numbers**

One thing to notice about PHP is that it provides automatic data type conversion.

So, if you assign an integer value to a variable, the type of that variable will automatically be an integer. Then, if you assign a string to the same variable, the type will change to a string.

This automatic conversion can sometimes break your code.

**PHP Integers**

2, 256, -256, 10358, -179567 are all integers.

An integer is a number without any decimal part.

An integer data type is a non-decimal number between -2147483648 and 2147483647 in 32 bit systems, and between -9223372036854775808 and 9223372036854775807 in 64 bit systems. A value greater (or lower) than this, will be stored as float, because it exceeds the limit of an integer.

**Note:** Another important thing to know is that even if 4 \* 2.5 is 10, the result is stored as float, because one of the operands is a float (2.5).

Here are some rules for integers:

* An integer must have at least one digit
* An integer must NOT have a decimal point
* An integer can be either positive or negative
* Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

PHP has the following predefined constants for integers:

* PHP\_INT\_MAX - The largest integer supported
* PHP\_INT\_MIN - The smallest integer supported
* PHP\_INT\_SIZE -  The size of an integer in bytes

PHP has the following functions to check if the type of a variable is integer:

* is\_int()
* is\_integer() - alias of is\_int()
* is\_long() - alias of is\_int()

**Example**

Check if the type of a variable is integer:

<?php   
$x = 5985;  
var\_dump(is\_int($x));  
  
$x = 59.85;  
var\_dump(is\_int($x));  
?>

**PHP Floats**

A float is a number with a decimal point or a number in exponential form.

2.0, 256.4, 10.358, 7.64E+5, 5.56E-5 are all floats.

The float data type can commonly store a value up to 1.7976931348623E+308 (platform dependent), and have a maximum precision of 14 digits.

PHP has the following predefined constants for floats (from PHP 7.2):

* PHP\_FLOAT\_MAX - The largest representable floating point number
* PHP\_FLOAT\_MIN - The smallest representable positive floating point number
* PHP\_FLOAT\_MAX - The smallest representable negative floating point number
* PHP\_FLOAT\_DIG - The number of decimal digits that can be rounded into a float and back without precision loss
* PHP\_FLOAT\_EPSILON - The smallest representable positive number x, so that x + 1.0 != 1.0

PHP has the following functions to check if the type of a variable is float:

* is\_float()
* is\_double() - alias of is\_float()

**Example**

Check if the type of a variable is float:

<?php   
$x = 10.365;  
var\_dump(is\_float($x));  
?>

**PHP Infinity**

A numeric value that is larger than PHP\_FLOAT\_MAX is considered infinite.

PHP has the following functions to check if a numeric value is finite or infinite:

* [is\_finite()](https://www.w3schools.com/php/func_math_is_finite.asp)
* [is\_infinite()](https://www.w3schools.com/php/func_math_is_infinite.asp)

However, the PHP var\_dump() function returns the data type and value:

**Example**

Check if a numeric value is finite or infinite:

<?php   
$x = 1.9e411;  
var\_dump($x);  
?>

**PHP NaN**

NaN stands for Not a Number.

NaN is used for impossible mathematical operations.

PHP has the following functions to check if a value is not a number:

* [is\_nan()](https://www.w3schools.com/php/func_math_is_nan.asp)

However, the PHP var\_dump() function returns the data type and value:

**Example**

Invalid calculation will return a NaN value:

<?php  
$x = acos(8);  
var\_dump($x);  
?>

**PHP Numerical Strings**

The PHP is\_numeric() function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise.

**Example**

Check if the variable is numeric:

<?php   
$x = 5985;  
var\_dump(is\_numeric($x));  
  
$x = "5985";  
var\_dump(is\_numeric($x));  
  
$x = "59.85" + 100;  
var\_dump(is\_numeric($x));  
  
$x = "Hello";  
var\_dump(is\_numeric($x));  
?>

**Note:** From PHP 7.0: The is\_numeric() function will return FALSE for numeric strings in hexadecimal form (e.g. 0xf4c3b00c), as they are no longer considered as numeric strings.

**PHP Casting Strings and Floats to Integers**

Sometimes you need to cast a numerical value into another data type.

The (int), (integer), or intval() function are often used to convert a value to an integer.

**Example**

Cast float and string to integer:

<?php  
// Cast float to int   
$x = 23465.768;  
$int\_cast = (int)$x;  
echo $int\_cast;  
  
echo "<br>";  
  
// Cast string to int  
$x = "23465.768";  
$int\_cast = (int)$x;  
echo $int\_cast;  
?>

PHP has a set of math functions that allows you to perform mathematical tasks on numbers.

## PHP pi() Function

The pi() function returns the value of PI:

### Example

<?php  
echo(pi()); // returns 3.1415926535898  
?>

## PHP min() and max() Functions

The min() and max() functions can be used to find the lowest or highest value in a list of arguments:

### Example

<?php  
echo(min(0, 150, 30, 20, -8, -200));  // returns -200  
echo(max(0, 150, 30, 20, -8, -200));  // returns 150  
?>

## PHP abs() Function

The abs() function returns the absolute (positive) value of a number:

### Example

<?php  
echo(abs(-6.7));  // returns 6.7  
?>

## PHP sqrt() Function

The sqrt() function returns the square root of a number:

### Example

<?php  
echo(sqrt(64));  // returns 8  
?>

## PHP round() Function

The round() function rounds a floating-point number to its nearest integer:

### Example

<?php  
echo(round(0.60));  // returns 1  
echo(round(0.49));  // returns 0  
?>

## Random Numbers

The rand() function generates a random number:

### Example

<?php  
echo(rand());  
?>

To get more control over the random number, you can add the optional min and max parameters to specify the lowest integer and the highest integer to be returned.

For example, if you want a random integer between 10 and 100 (inclusive), use rand(10, 100):

### Example

<?php  
echo(rand(10, 100));  
?>

**PHP Constants**

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.

A valid constant name starts with a letter or underscore (no $ sign before the constant name).

**Note:** Unlike variables, constants are automatically global across the entire script.

**Create a PHP Constant**

To create a constant, use the define() function.

**Syntax**

define(*name*, *value*, *case-insensitive*)

Parameters:

* *name*: Specifies the name of the constant
* *value*: Specifies the value of the constant
* *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

**Example**

Create a constant with a **case-sensitive** name:

<?php  
define("GREETING", "Welcome to W3Schools.com!");  
echo GREETING;  
?>

**Example**

Create a constant with a **case-insensitive** name:

<?php  
define("GREETING", "Welcome to W3Schools.com!", true);  
echo greeting;  
?>

**PHP Constant Arrays**

In PHP7, you can create an Array constant using the define() function.

**Example**

Create an Array constant:

<?php  
define("cars", [  
  "Alfa Romeo",  
  "BMW",  
  "Toyota"  
]);  
echo cars[0];  
?>

**Constants are Global**

Constants are automatically global and can be used across the entire script.

**Example**

This example uses a constant inside a function, even if it is defined outside the function:

<?php  
define("GREETING", "Welcome to W3Schools.com!");  
  
function myTest() {  
  echo GREETING;  
}  
   
myTest();  
?>

**PHP Operators**

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Increment/Decrement operators
* Logical operators
* String operators
* Array operators
* Conditional assignment operators

**PHP Arithmetic Operators**

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** | **Show it** |
| + | Addition | $x + $y | Sum of $x and $y |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| - | Subtraction | $x - $y | Difference of $x and $y |  |
| \* | Multiplication | $x \* $y | Product of $x and $y |  |
| / | Division | $x / $y | Quotient of $x and $y |  |
| % | Modulus | $x % $y | Remainder of $x divided by $y |  |
| \*\* | Exponentiation | $x \*\* $y | Result of raising $x to the $y'th power |  |

**PHP Assignment Operators**

The PHP assignment operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

|  |  |  |  |
| --- | --- | --- | --- |
| **Assignment** | **Same as...** | **Description** | **Show it** |
| x = y | x = y | The left operand gets set to the value of the expression on the right |  |

|  |  |  |  |
| --- | --- | --- | --- |
| x += y | x = x + y | Addition |  |
| x -= y | x = x - y | Subtraction |  |
| x \*= y | x = x \* y | Multiplication |  |
| x /= y | x = x / y | Division |  |
| x %= y | x = x % y | Modulus |  |

**PHP Comparison Operators**

The PHP comparison operators are used to compare two values (number or string):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** | **Show it** |
| == | Equal | $x == $y | Returns true if $x is equal to $y |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| === | Identical | $x === $y | Returns true if $x is equal to $y, and they are of the same type |  |
| != | Not equal | $x != $y | Returns true if $x is not equal to $y |  |
| <> | Not equal | $x <> $y | Returns true if $x is not equal to $y |  |
| !== | Not identical | $x !== $y | Returns true if $x is not equal to $y, or they are not of the same type |  |
| > | Greater than | $x > $y | Returns true if $x is greater than $y |  |
| < | Less than | $x < $y | Returns true if $x is less than $y |  |
| >= | Greater than or equal to | $x >= $y | Returns true if $x is greater than or equal to $y |  |
| <= | Less than or equal to | $x <= $y | Returns true if $x is less than or equal to $y |  |
| <=> | Spaceship | $x <=> $y | Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7. |  |

**PHP Increment / Decrement Operators**

The PHP increment operators are used to increment a variable's value.

The PHP decrement operators are used to decrement a variable's value.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Show it** |
| ++$x | Pre-increment | Increments $x by one, then returns $x |  |

|  |  |  |  |
| --- | --- | --- | --- |
| $x++ | Post-increment | Returns $x, then increments $x by one |  |
| --$x | Pre-decrement | Decrements $x by one, then returns $x |  |
| $x-- | Post-decrement | Returns $x, then decrements $x by one |  |

**PHP Logical Operators**

The PHP logical operators are used to combine conditional statements.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** | **Show it** |
| and | And | $x and $y | True if both $x and $y are true |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| or | Or | $x or $y | True if either $x or $y is true |  |
| xor | Xor | $x xor $y | True if either $x or $y is true, but not both |  |
| && | And | $x && $y | True if both $x and $y are true |  |
| || | Or | $x || $y | True if either $x or $y is true |  |
| ! | Not | !$x | True if $x is not true |  |

**PHP String Operators**

PHP has two operators that are specially designed for strings.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** | **Show it** |
| . | Concatenation | $txt1 . $txt2 | Concatenation of $txt1 and $txt2 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| .= | Concatenation assignment | $txt1 .= $txt2 | Appends $txt2 to $txt1 |  |

**PHP Array Operators**

The PHP array operators are used to compare arrays.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** | **Show it** |
| + | Union | $x + $y | Union of $x and $y |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| == | Equality | $x == $y | Returns true if $x and $y have the same key/value pairs |  |
| === | Identity | $x === $y | Returns true if $x and $y have the same key/value pairs in the same order and of the same types |  |
| != | Inequality | $x != $y | Returns true if $x is not equal to $y |  |
| <> | Inequality | $x <> $y | Returns true if $x is not equal to $y |  |
| !== | Non-identity | $x !== $y | Returns true if $x is not identical to $y |  |

**PHP Conditional Assignment Operators**

The PHP conditional assignment operators are used to set a value depending on conditions:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** | **Show it** |
| ?: | Ternary | $x = *expr1* ? *expr2* : *expr3* | Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ?? | Null coalescing | $x = *expr1* ?? *expr2* | Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7 |  |

[[1]](#footnote-1)

1. **PHP Conditional Statements**

   Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

   In PHP we have the following conditional statements:

   * if statement - executes some code if one condition is true
   * if...else statement - executes some code if a condition is true and another code if that condition is false
   * if...elseif...else statement - executes different codes for more than two conditions
   * switch statement - selects one of many blocks of code to be executed

   **PHP - The if Statement**

   The if statement executes some code if one condition is true.

   **Syntax**

   if (*condition*) { *code to be executed if condition is true*;  
   }

   **Example**

   Output "Have a good day!" if the current time (HOUR) is less than 20:

   <?php  
   $t = date("H");  
     
   if ($t < "20") {  
     echo "Have a good day!";  
   }  
   ?>

   **PHP - The if...else Statement**

   The if...else statement executes some code if a condition is true and another code if that condition is false.

   **Syntax**

   if (*condition*) {  
     *code to be executed if condition is true;*  
   } else {  
     *code to be executed if condition is false;*}

   **Example**

   Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

   <?php  
   $t = date("H");  
     
   if ($t < "20") {  
     echo "Have a good day!";  
   } else {  
     echo "Have a good night!";  
   }  
   ?>

   **PHP - The if...elseif...else Statement**

   The if...elseif...else statement executes different codes for more than two conditions.

   **Syntax**

   if (*condition*) {  
     *code to be executed if this condition is true;*} elseif (*condition*) {  
     *code to be executed if first condition is false and this condition is true;*} else {  
     *code to be executed if all conditions are false;*}

   **Example**

   Output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

   <?php  
   $t = date("H");  
     
   if ($t < "10") {  
     echo "Have a good morning!";  
   } elseif ($t < "20") {  
     echo "Have a good day!";  
   } else {  
     echo "Have a good night!";  
   }  
   ?>

   **PHP - The switch Statement**

   ## The PHP switch Statement

   Use the switch statement to **select one of many blocks of code to be executed**.

   ### Syntax

   switch (*n*) {  
     case *label1:*  
     *code to be executed if n=label1;*  
       break;  
     case *label2:*  
     *code to be executed if n=label2;*  
       break;  
     case *label3:*  
     *code to be executed if n=label3;*  
       break;  
       ...  
     default:  
     *code to be executed if n is different from all labels;*  
   }

   This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically. The default statement is used if no match is found.

   ### Example

   <?php  
   $favcolor = "red";  
     
   switch ($favcolor) {  
     case "red":  
       echo "Your favorite color is red!";  
       break;  
     case "blue":  
       echo "Your favorite color is blue!";  
       break;  
     case "green":  
       echo "Your favorite color is green!";  
       break;  
     default:  
       echo "Your favorite color is neither red, blue, nor green!";  
   }  
   ?>

   **PHP Loops**

   Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

   Loops are used to execute the same block of code again and again, as long as a certain condition is true.

   In PHP, we have the following loop types:

   * while - loops through a block of code as long as the specified condition is true
   * do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
   * for - loops through a block of code a specified number of times
   * foreach - loops through a block of code for each element in an array

   The following chapters will explain and give examples of each loop type.

   **The PHP while Loop**

   The while loop executes a block of code as long as the specified condition is true.

   **Syntax**

   while (*condition is true*) {  
   *code to be executed*;  
   }

   **Examples**

   The example below displays the numbers from 1 to 5:

   **Example**

   <?php   
   $x = 1;   
     
   while($x <= 5) {  
     echo "The number is: $x <br>";  
     $x++;  
   }   
   ?>

   **Example Explained**

   * $x = 1; - Initialize the loop counter ($x), and set the start value to 1
   * $x <= 5 - Continue the loop as long as $x is less than or equal to 5
   * $x++; - Increase the loop counter value by 1 for each iteration

   This example counts to 100 by tens:

   **Example**

   <?php   
   $x = 0;   
     
   while($x <= 100) {  
     echo "The number is: $x <br>";  
     $x+=10;  
   }   
   ?>

   **Example Explained**

   * $x = 0; - Initialize the loop counter ($x), and set the start value to 0
   * $x <= 100 - Continue the loop as long as $x is less than or equal to 100
   * $x+=10; - Increase the loop counter value by 10 for each iteration

   ## GET vs. POST

   Both GET and POST create an array (e.g. array( key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

   Both GET and POST are treated as $\_GET and $\_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

   $\_GET is an array of variables passed to the current script via the URL parameters.

   $\_POST is an array of variables passed to the current script via the HTTP POST method.

   ## When to use GET?

   Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

   GET may be used for sending non-sensitive data.

   **Note:** GET should NEVER be used for sending passwords or other sensitive information!

   ## When to use POST?

   Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

   Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

   However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

   **Developers prefer POST for sending form data.** [↑](#footnote-ref-1)